# Reporting an Issue

- Report all issues

- Trivial documentation issues welcome ([example](https://github.com/bwapi/bwapi/issues/616))

- Non-trivial feature requests welcome ([example](https://github.com/bwapi/bwapi/issues/393))

# Creating a Pull Request

- Do development on the [`develop`](https://github.com/bwapi/bwapi/tree/develop) branch.

# Versioning

BWAPI versioning is categorized as follows: `major.minor.patch [Beta]`

Version Component | Description

------------------|----------------

major | Increased when massive structural changes are made.

minor | Increased when breaking changes are made. Modules will need to be recompiled.

patch | Increased when non-breaking changes are made.

Beta | Appended to a major version increase until stability has been verified.

# Coding Standards

## Spacing

- Use double-spaces instead of tabs. You should be able to convert tabs to spaces automatically in your editor's settings.

- Always indent for each scope level.

- Always put a space between all arithmetic, binary, and conditional operators.

- Don't put a space between function names and the open parenthesis.

- Put a space after a comma.

Examples of spacing

~~~{.cpp}

// bad, can't tell if assignment is a typo, and

// there is a space between the member function and its parameters

if(Broodwar->canMake (UnitTypes::Terran\_Marine, builder))

reservedMinerals=-UnitTypes::Terran\_Marine.mineralPrice();

// good, the arithmetic is clear

if(Broodwar->canMake(UnitTypes::Terran\_Marine, builder))

reservedMinerals = -UnitTypes::Terran\_Marine.mineralPrice();

// bad, the scope is not indented, no space following the comma

if ( Broodwar->canMake(UnitTypes::Terran\_Marine,builder) )

reservedMinerals = -UnitTypes::Terran\_Marine.mineralPrice();

// good, alternative spacing style for if statements

if ( Broodwar->canMake(UnitTypes::Terran\_Marine, builder) )

reservedMinerals = -UnitTypes::Terran\_Marine.mineralPrice();

~~~

## Formatting

- Use [ANSI style](http://en.wikipedia.org/wiki/Indent\_style#Allman\_style) braces.

- Break up large single-line if statements to be multi-line if statements.

- Format constructor initializer lists as follows (so that the colon is in the same column as the commas):

~~~{.cpp}

BulletImpl::BulletImpl(BW::CBullet\* originalBullet, u16 \_index)

: bwOriginalBullet(originalBullet)

, index(\_index)

{

}

~~~

## Naming

- Use meaningful names. If a variable's purpose cannot be identified by another project member without analysing the code, then the variable needs to be renamed.

- In local looping scopes, single-letter variable names are generally used as follows:

- `b` for Bullet

- `f` for Force

- `p` for Player

- `r` for Region

- `u` for Unit

- `i` for iterator/index

- Member variables and member function names should be in [lower camel case](http://en.wikipedia.org/wiki/CamelCase). Examples: `getUnitsInRectangle`, `wasSeenByBWAPIPlayer`.

- Constants and macros should be in ALL CAPS and words separated by underscores. Example: `PLAYER\_COUNT`

- Use [upper Camel Case](http://en.wikipedia.org/wiki/CamelCase) for classes, structures, enums, and namespaces.

## Hacking

- Offsets must be in hexadecimal. Example: `0x00408CF0`.

- Offsets must be placed in `BW/offsets.h`.

- Avoid using inline assembly unless it is impossible to do so.

- Always perform version checking before making code patches, to maintain partial cross-version compatibility.

## Documentation

- Use the triple slash (`///`) format.

- Try to include as much details on the function as possible. Redundancy can express clarity.

- Include example code if possible.

- Wrap lines before the 100th column.

### Doxygen Generation

- Use `@` for doxygen commands.

- Use `@see` to refer to other functions/classes.

- When introducing a new function, include a `@since` tag with the version number.

- Include `@returns`, and `@retval` where necessary.

- When specifying `@param`, also specify if the argument is optional. Present an indented description on the next line, and also specify its default value if applicable.

### Intellisense Compatibility

- Use the [summary tag](https://msdn.microsoft.com/en-us/library/ms177242.aspx) to wrap the function/class/enum description.

- Use the [param tag](https://msdn.microsoft.com/en-us/library/ms177235.aspx) to identify all of the parameters.

- Identify if an argument is optional with `(optional)` on the same line as the tag.

- Put parameter descriptions indented on the next line, between the tags.

Examples of documentation:

~~~{.cpp}

/// <summary>Sets the size of the text for all calls to drawText following this one.</summary>

///

/// <param name="size"> (optional)

/// The size of the text. This value is one of Text::Size::Enum. If this value is omitted,

/// then a default value of Text::Size::Default is used.

/// </param>

///

/// Example usage

/// @code

/// void ExampleAIModule::onFrame()

/// {

/// // Centers the name of the player in the upper middle of the screen

/// BWAPI::Broodwar->setTextSize(BWAPI::Text::Size::Large);

/// BWAPI::Broodwar->drawTextScreen(BWAPI::Positions::Origin, "%c%c%s",

/// BWAPI::Text::Align\_Center,

/// BWAPI::Text::Green,

/// BWAPI::Broodwar->self()->getName().c\_str() );

/// BWAPI::Broodwar->setTextSize(); // Set text size back to default

/// }

/// @endcode

/// @see Text::Size::Enum

virtual void setTextSize(Text::Size::Enum size = Text::Size::Default) = 0;

~~~

~~~{.cpp}

/// <summary>Retrieves the region at a given position.</summary>

///

/// <param name="x">

/// The x coordinate, in pixels.

/// </param>

/// <param name="y">

/// The y coordinate, in pixels.

/// </param>

///

/// @returns Pointer to the Region interface at the given position.

/// @retval nullptr if the provided position is not valid (i.e. not within the map bounds).

///

/// @note If the provided position is invalid, the error Errors::Invalid\_Parameter is set.

/// @see getAllRegions, getRegion

virtual BWAPI::Region getRegionAt(int x, int y) const = 0;

/// @overload

BWAPI::Region getRegionAt(BWAPI::Position position) const;

~~~

## Language Features

- Use the `nullptr` keyword instead of `NULL`.

- Create move constructors if appropriate.

- Use the `const` keyword where appropriate.

- Use std::array instead of C-style arrays.

- Use explicit enum types (preferably enum class) instead of ints.

- Use in-class member initialization.

- Use the keywords default, delete, override, final, etc.

# Changing the API

## Adding a Virtual Function

1. Add it after all other virtual functions.

2. Label it with Doxygen `@since` tag, indicating the version it was introduced.

Note: Virtual functions are implementation defined, but Visual Studio appears to maintain some consistency regarding the use of virtual functions. Adding a new function to the end will maintain some backwards compatibility.

## Renaming a Function

1. Rename the function.

2. Create a non-virtual function with the old name that calls the new function.

3. Label it with Doxygen `@deprecated` tag and refer to the new function.

4. Optionally add a compiler deprecation warning (until we move to VS 2015).

5. Remove it after the next 2 minor versions or next major version, whichever comes first.

## Deprecating a Function

1. Label it with Doxygen `@deprecated` tag.

2. Provide reason for the deprecation and alternatives if applicable.

3. Optionally add a compiler deprecation warning (until we move to VS 2015).

4. Remove it after the next 2 minor versions or next major version, whichever comes first.